java.net.InetAddress

java.net.Inet4Address

java.net.Inet6Address

java.net.URL

java.net.Socket (TCP client),

java.net.ServerSocket (TCP server),

java.net.DatagramSocket (UDP client and server),

javax.net.ssl.SSLSocket (SSL client),

javax.net.ssl.SSLServerSocket (SSL server)

Parsing a URL

The `URL` class provides several methods that let you query `URL` objects. You can get the protocol, host name, port number, and filename from a URL using these accessor methods:

`getProtocol`
: Returns the protocol identifier component of the URL.

`getHost`
: Returns the host name component of the URL.

`getPort`
: Returns the port number component of the URL. The `getPort` method returns an integer that is the port number. If the port is not set, `getPort` returns -1.

`getFile`
: Returns the filename component of the URL.

`getRef`
: Returns the reference component of the URL.

```
import java.net.*;
import java.io.*;

public class ParseURL {
    public static void main(String[] args) throws Exception {
        URL aURL = new URL("http://java.sun.com:80/docs/books/"
                           + "tutorial/index.html#DOWNLOADING");

        System.out.println("protocol = " + aURL.getProtocol());
        System.out.println("host = " + aURL.getHost());
        System.out.println("filename = " + aURL.getFile());
        System.out.println("port = " + aURL.getPort());
        System.out.println("ref = " + aURL.getRef());
    }
}
```

Reading directly from URL

The following small Java program uses `openStream()` to get an input stream on the URL `http://www.yahoo.com/`. It then opens a `BufferedReader` on the input stream and reads from the `BufferedReader` thereby reading from the URL. Everything read is copied to the standard output stream:

```
import java.net.*;
import java.io.*;

public class URLReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                yahoo.openStream()));

        String inputLine;

        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);

        in.close();
    }
}
```

Connecting to a URL

After you've successfully created a `URL` object, you can call the `URL` object's `openConnection` method to connect to it. When you connect to a `URL`, you are initializing a communication link between your Java program and the URL over the network. For example, you can open a connection to the Yahoo site with the following code:

```
try {
    URL yahoo = new URL("http://www.yahoo.com/");
    URLConnection yahooConnection = yahoo.openConnection();

} catch (MalformedURLException e) {     // new URL() failed
    . . .
} catch (IOException e) {               // openConnection() failed
    . . .
}
```

Reading from a URLConnection

The following program performs the same function as the `URLReader` program shown in Reading Directly from a URL.

However, rather than getting an input stream directly from the URL, this program explicitly opens a connection to a URL and gets an input stream from the connection. Then, like `URLReader`, this program creates a `BufferedReader` on the input stream and reads from it. The bold statements highlight the differences between this example and the previous

```
import java.net.*;
import java.io.*;

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                yc.getInputStream()));
        String inputLine;

        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Writing to a URLConnection

Many HTML pages contain *forms*-- text fields and other GUI objects that let you enter data to send to the server. After you type in the required information and initiate the query by clicking a button, your Web browser writes the data to the URL over the network. At the other end, a `cgi-bin` script (usually) on the server receives the data, processes it, and then sends you a response, usually in the form of a new HTML page.

Here's an example program that runs the backwards script over the network through a `URLConnection`:

```
import java.io.*;
import java.net.*;

public class Reverse {
    public static void main(String[] args) throws Exception {

     if (args.length != 1) {
         System.err.println("Usage:  java Reverse "
                            + "string_to_reverse");
         System.exit(1);
     }

     String stringToReverse = URLEncoder.encode(args[0]);

     URL url = new URL("http://java.sun.com/cgi-bin/backwards");
     URLConnection connection = url.openConnection();
     connection.setDoOutput(true);

     PrintWriter out = new PrintWriter(
                               connection.getOutputStream());
     out.println("string=" + stringToReverse);
     out.close();

     BufferedReader in = new BufferedReader(
                             new InputStreamReader(
                             connection.getInputStream()));
     String inputLine;

     while ((inputLine = in.readLine()) != null)
         System.out.println(inputLine);

     in.close();
    }
}
```

Reading from and Writing to a Socket

`EchoClient` creates a socket thereby getting a connection to the Echo server. It reads input from the user on the standard input stream, and then forwards that text to the Echo server by writing the text to the socket. The server echoes the input back through the socket to the client. The client program reads and displays the data passed back to it from the server:

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {

        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        try {
            echoSocket = new Socket("taranis", 7);
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(
                                    echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: taranis.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for "
                               + "the connection to: taranis.");
            System.exit(1);
        }

        BufferedReader stdIn = new BufferedReader(
                                   new InputStreamReader(System.in));
        String userInput;

        while ((userInput = stdIn.readLine()) != null) {
            out.println(userInput);
            System.out.println("echo: " + in.readLine());
        }

        out.close();
        in.close();
        stdIn.close();
        echoSocket.close();
    }
}
```

Writing the Server Side of a Socket

```
import java.net.*;
import java.io.*;

public class EchoServer {
    public static void main(String[] args) throws IOException {

        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(4444);
        } catch (IOException e) {
            System.err.println("Could not listen on port: 4444.");
            System.exit(1);
        }

        Socket clientSocket = null;
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept failed.");
            System.exit(1);
        }

        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                clientSocket.getInputStream()));
        String inputLine, outputLine;

        while ((inputLine = in.readLine()) != null) {
             outputLine = inputLine;
             out.println(outputLine);
             if (outputLine.equals("Bye."))
                break;
        }
        out.close();
        in.close();
        clientSocket.close();
        serverSocket.close();
    }
}
```

Threaded Server Pattern

```
ServerSocket server = new ServerSocket(port);

While(true) {
  Socket socket = server.accept();
  Handler handler = new Handler(socket);
  Handler.start();
}
public class handler extend Thread {

protected final Socket socket;
public Handler (Socket socket) {
   this.socket = socket;
}

public void run() {
   try {


   } finally {
      Socket.close();
   }
}
}
```

Writing the MultiServer

```
import java.net.*;
import java.io.*;

public class MultiServer {

   final static short port = 4444;
   public static void main(String[] args) throws IOException {ServerSocket serverSocket = null;
      boolean doomsday = true;

      try {
         System.out.println("New server");
         serverSocket = new ServerSocket(port);
      } catch (IOException e) {
         System.err.println("Could not listen on port: "+port+"."); System.exit(-1);
      }

      while (doomsday) new MultiServerThread(serverSocket.accept()).start();

      serverSocket.close();
   }
}

class MultiServerThread extends Thread {
   private Socket socket = null;

   public MultiServerThread(Socket socket) {
         super("MultiServerThread"); this.socket = socket;
   }

   public void run() {
         try {
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader( new InputStreamReader( socket.getInputStream()));

            String inputLine, outputLine;

            while ((inputLine = in.readLine()) != null) {
                  outputLine = inputLine;
                  out.println(outputLine); if (outputLine.equals("Bye")) break;
            }
            out.close(); in.close(); socket.close();

         } catch (IOException e) {
            e.printStackTrace();
         }
   }
}
```

Datagram Socket, Datagram packet, Multicast Socket

```
import java.io.*;
import java.net.*;
import java.util.*;

public class QuoteServer {
   public static void main(String[] args) throws IOException {
      new QuoteServerThread().start();
   }
}

class QuoteServerThread extends Thread {

   protected DatagramSocket socket = null;
   protected BufferedReader in = null;
   protected boolean moreQuotes = true;

   public QuoteServerThread() throws IOException {
         this("QuoteServerThread");
   }

   public QuoteServerThread(String name) throws IOException {
      super(name);
      socket = new DatagramSocket(4445);

      try {
         in = new BufferedReader(new FileReader("one-liners.txt"));
      } catch (FileNotFoundException e) {
         System.err.println("Could not open quote file. Serving time instead.");
      }
   }

   public void run() {

      while (moreQuotes) {
         try {
            byte[] buf = new byte[256];

               // receive request
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);

               // figure out response
            String dString = null;
            if (in == null)
               dString = new Date().toString();
            else
               dString = getNextQuote();
            buf = dString.getBytes();
```

```
                    // send the response to the client at "address" and "port"
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(buf, buf.length, address, port);
            socket.send(packet);
          } catch (IOException e) {
            e.printStackTrace();
                  moreQuotes = false;
          }
        }
        socket.close();
    }

    protected String getNextQuote() {
        String returnValue = null;
        try {
          if ((returnValue = in.readLine()) == null) {
            in.close();
                  moreQuotes = false;
            returnValue = "No more quotes. Goodbye.";
          }
        } catch (IOException e) {
          returnValue = "IOException occurred in server.";
        }
        return returnValue;
    }
}
```

Multicast Datagram Server

```
import java.io.*;
import java.net.*;
import java.util.*;

public class MulticastServer {
   public static void main(String[] args) throws java.io.IOException {
      new MulticastServerThread().start();
   }
}

class MulticastServerThread extends QuoteServerThread {

   private long FIVE_SECONDS = 5000;

   final static short port = 4446;
   final static String m_addr = "230.0.0.1";

   public MulticastServerThread() throws IOException {
      super("MulticastServerThread");
   }

   public void run() {
      while (moreQuotes) {
         try {
            byte[] buf = new byte[256];

               // construct quote
            String dString = null;
            if (in == null)
               dString = new Date().toString();
            else
               dString = getNextQuote();
            buf = dString.getBytes();

                  // send it
            InetAddress group = InetAddress.getByName(m_addr);
            DatagramPacket packet = new DatagramPacket(buf, buf.length, group, port);
            socket.send(packet);

                  // sleep for a while
                try {
                   sleep((long)(Math.random() * FIVE_SECONDS));
                } catch (InterruptedException e) { }
         } catch (IOException e) {
            e.printStackTrace();
                  moreQuotes = false;
         }
```

```
        }
            socket.close();
    }
}
```

Multicast Datagram Client

```
import java.io.*;
import java.net.*;
import java.util.*;

public class MulticastClient {
   final static short port = 4446;
   final static String m_addr = "230.0.0.1";

   public static void main(String[] args) throws IOException {

      MulticastSocket socket = new MulticastSocket(port);
      InetAddress address = InetAddress.getByName(m_addr);
         socket.joinGroup(address);

      DatagramPacket packet;

         // get a few quotes
         for (int i = 0; i < 5; i++) {

            byte[] buf = new byte[256];
         packet = new DatagramPacket(buf, buf.length);
         socket.receive(packet);

         String received = new String(packet.getData());
         System.out.println("Quote of the Moment: " + received);
         }

         socket.leaveGroup(address);
         socket.close();
   }

}
```

Broadcast datagram

```
DatagramSocket s = new DatagramSocket(null);
s.bind(new InetSocketAddress(8888));
```

Which is equivalent to:

```
DatagramSocket s = new DatagramSocket(8888);
```

Both cases will create a DatagramSocket able to receive broadcasts on UDP port 8888.

Multicast Datagram Server

```
import java.io.*;
import java.net.*;

public class MulticastServer {


  public static void main(String[] args) throws Exception {

    InetAddress group = InetAddress.getByName("239.203.2.13");


    MulticastSocket socket = new MulticastSocket(4422);

    socket.joinGroup(group);

    byte[] buffer = new byte[512];

    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

    while(true) {

      socket.receive(packet);

      System.out.println(new java.util.Date() + ": " +
                                 new String(packet.getData(), 0, packet.getLength()));
      packet.setLength(buffer.length);
    }
  }
}
```

Multicast Datagram Client

```
import java.io.*;
import java.net.*;

public class MulticastClient {


  public static void main(String[] args) throws Exception {

     InetAddress group = InetAddress.getByName("239.203.2.13");

    MulticastSocket socket = new MulticastSocket();

    socket.setTimeToLive(1);

    byte[] data = "hello world".getBytes();

    DatagramPacket packet = new DatagramPacket
      (data, data.length, group, 4422);

    socket.send(packet);

    data = "followup call".getBytes();
    packet.setData(data);

    socket.send(packet);
  }
}
```